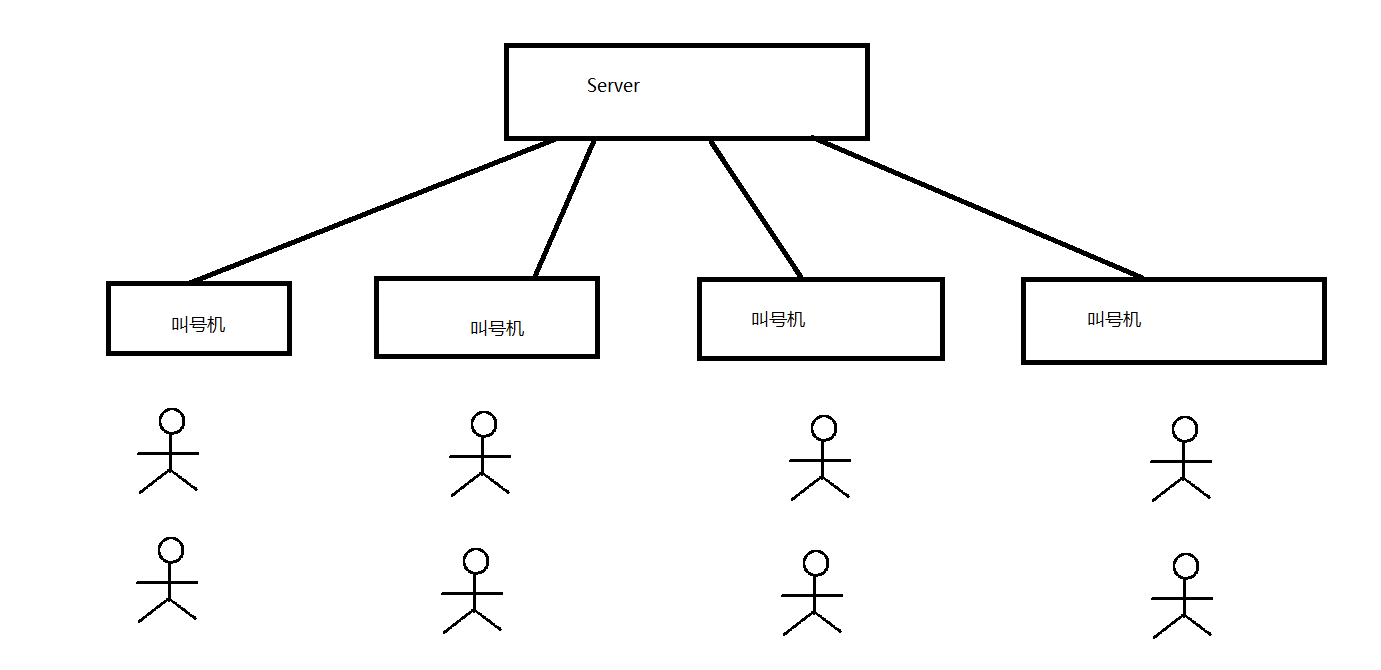
# Synchronized

并发叫号：



Static:并发量比较大的时候会出现：跳号、重号、超过最大值。

## 一、概念

Synchronized是利用锁的机制来实现同步的。

锁机制有如下两种特性：

**互斥性：**即在同一时间只允许一个线程持有某个对象锁，通过这种特性来实现多线程中的协调机制，这样在同一时间只有一个线程对需同步的代码块(复合操作)进行访问。互斥性我们也往往称为操作的原子性。

**可见性：**必须确保在锁被释放之前，对共享变量所做的修改，对于随后获得该锁的另一个线程是可见的（即在获得锁时应获得最新共享变量的值），否则另一个线程可能是在本地缓存的某个副本上继续操作从而引起不一致。

## **二、synchronized的用法**

**根据修饰对象分类**

1. 同步方法
   1. 同步非静态方法

Public synchronized void methodName(){

……

}

* 1. 同步静态方法

Public synchronized static void methodName(){

……

}

1. 同步代码块

**synchronized(this|object) {}**

**synchronized(类.class) {}**

Private final Object MUTEX =new Object();

Public void methodName(){

Synchronized(MUTEX ){

……

}

}

根据获取的锁分类

1. 获取对象锁

synchronized(this|object) {}

修饰非静态方法

在 Java 中，每个对象都会有一个 monitor 对象，这个对象其实就是 Java 对象的锁，通常会被称为“内置锁”或“对象锁”。类的对象可以有多个，所以每个对象有其独立的对象锁，互不干扰。

1. 获取类锁

synchronized(类.class) {}

修饰静态方法

在 Java 中，针对每个类也有一个锁，可以称为“类锁”，类锁实际上是通过对象锁实现的，即类的 Class 对象锁。每个类只有一个 Class 对象，所以每个类只有一个类锁。

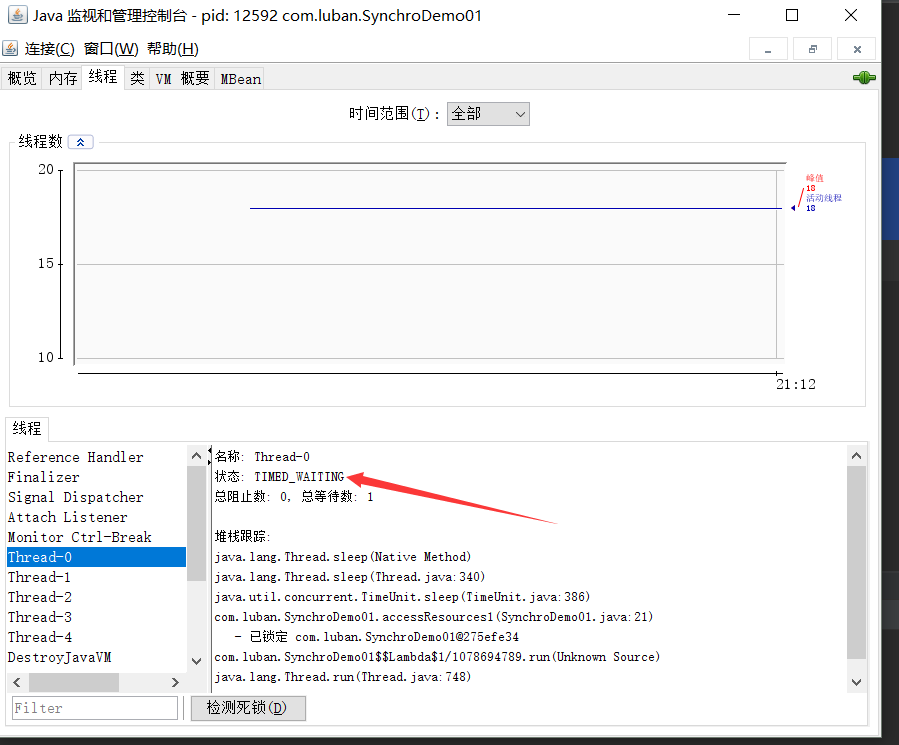
在 Java 中，每个对象都会有一个 monitor 对象，监视器。

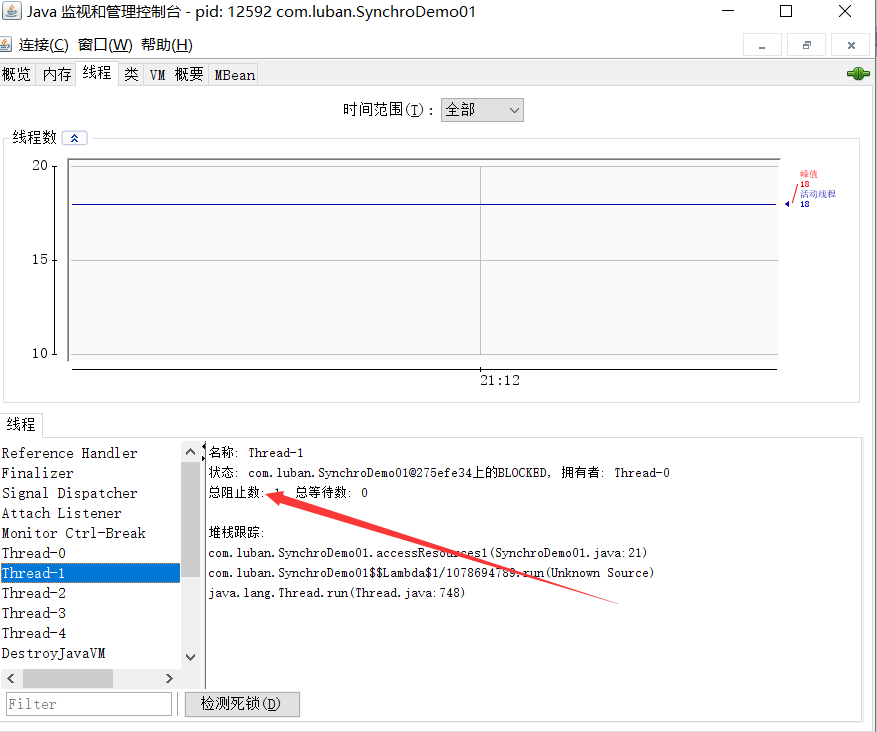
1. 某一线程占有这个对象的时候，先monitor 的计数器是不是0，如果是0还没有线程占有，这个时候线程占有这个对象，并且对这个对象的monitor+1；如果不为0，表示这个线程已经被其他线程占有，这个线程等待。当线程释放占有权的时候，monitor-1；
2. 同一线程可以对同一对象进行多次加锁，monitor +1再+1，具有重入性。

## **synchronized原理分析**

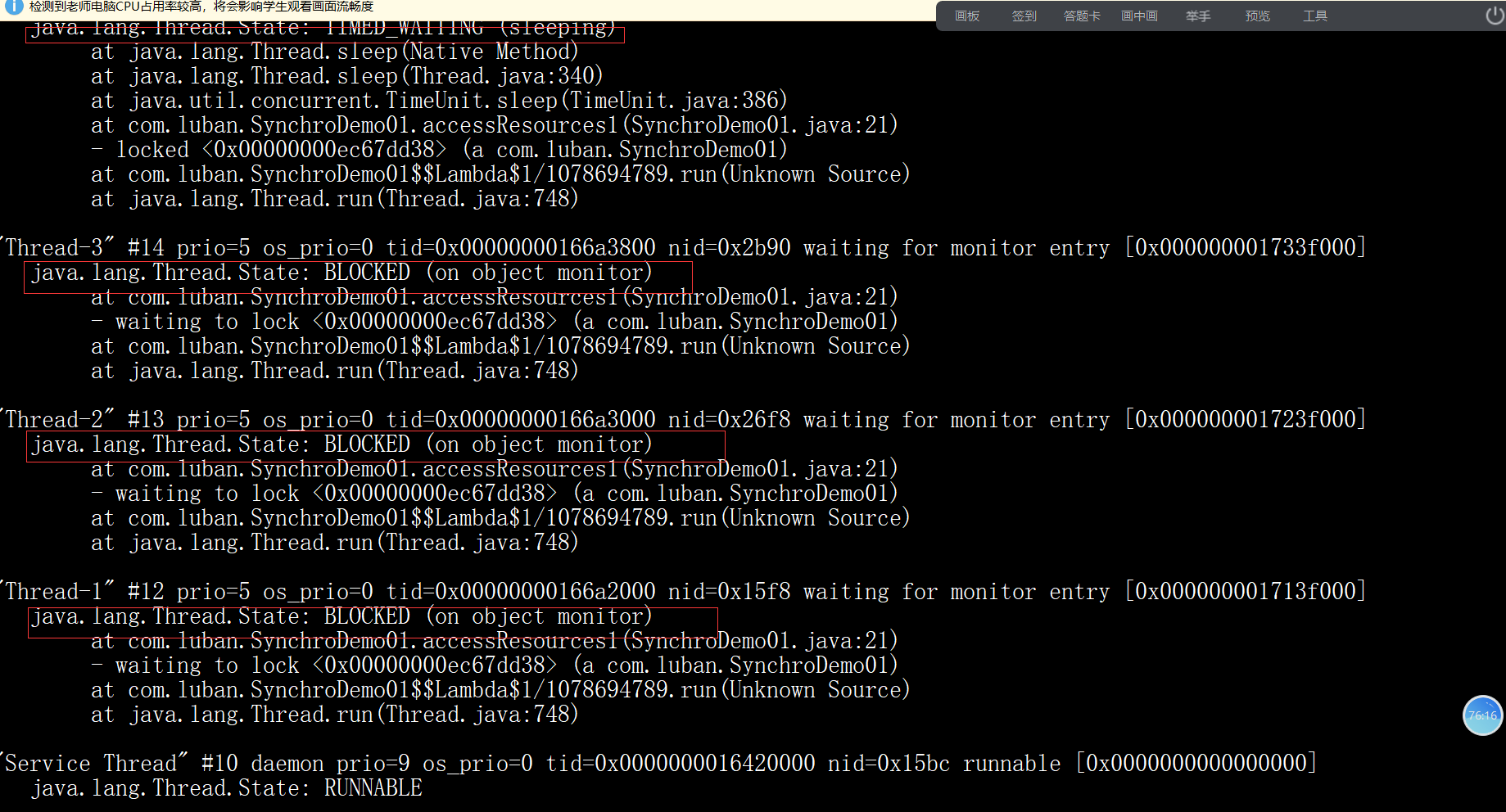
1. 线程堆栈分析（互斥）

Jconsole





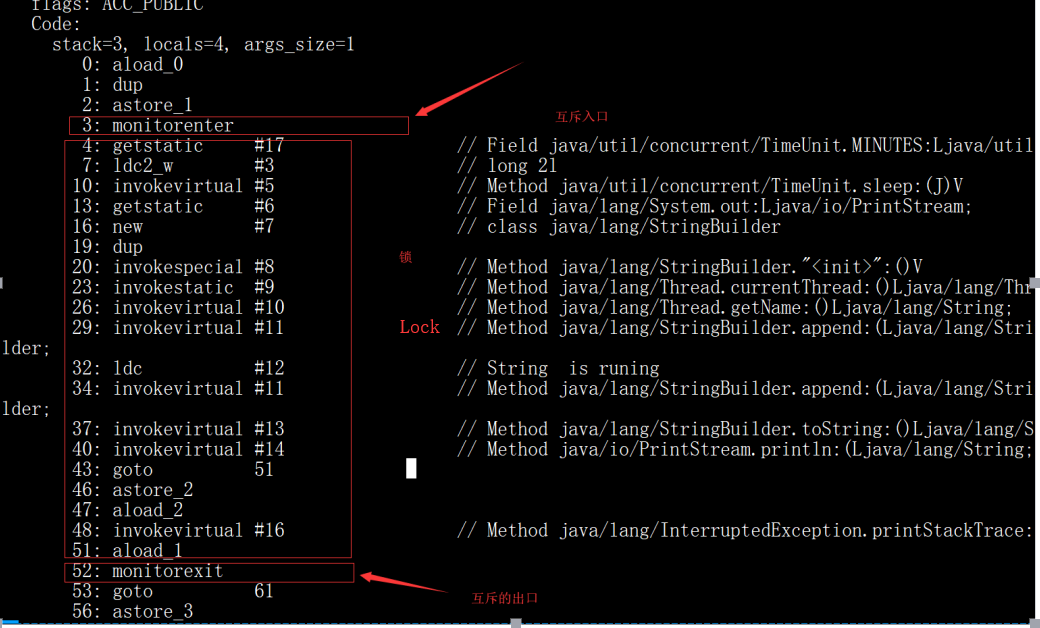
Jstack pid #查看



1. JVM指令分析

Javap -V 编译后的Class文件路径 #反编译

Monitorenter 与 Monitorexit

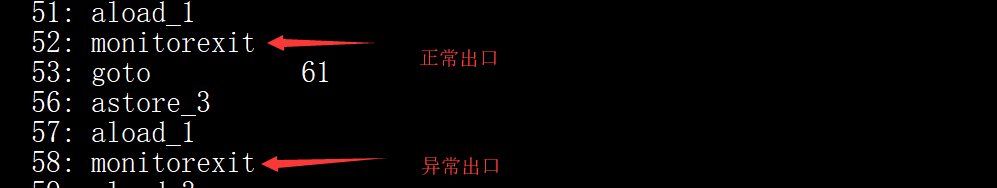


Monitor

1. 0，lock
2. 可重入性
3. monitor一个线程占有，其他线程请求时会进入BOLCK,直到monitor为0

Monitorexit

计数器减一，为0时为解锁



以上是代码块的加锁monitorenter和monitorExit配合使用

对方法的加锁

ACC\_SYNCHRONIZED



1.6JDK以前重量锁

1. 使用synchronized注意的问题
2. 与monitor关联的对象不能为空
3. synchronized作用域不能太大
4. 不同的monitor企图锁相同的方法
5. 多个锁的交叉导致死锁

## **Java虚拟机对synchronized的优化**

偏向锁

轻量级锁

重量级锁（等待时间长）

对象头与monitor

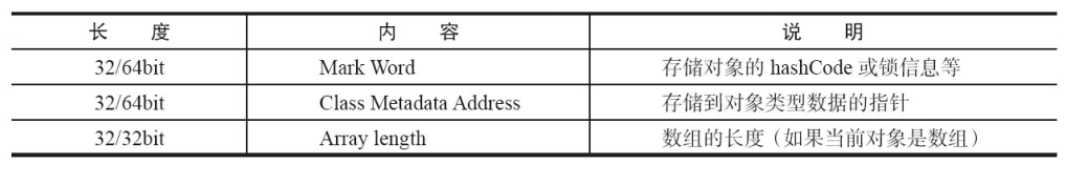
一个对象实例包含：对象头、实例变量、填充数据；



对象头：加锁的基础；

实例变量：

填充数据：





无锁状态:没有加锁

偏向锁：在对象第一次被某一线程占有的时候，是否偏向锁置1，锁表01，写入线程号，当其他的线 程访问的时候，竞争，失败进入轻量级锁；

很多时候第一次占有它的线程获取次数多，成功；

CAS算法 campare and set（CAS）无锁状态时间非常接近,竞争不激烈的时候适用；

轻量级锁：线程有交替适用，互斥性不是很强，CAS失败进入重量级锁，锁标记00

重量级锁：强互斥，锁标记10，等待时间长；

锁升级带来的是性能问题；

解决方案：

自旋锁：竞争失败的时候，不是马上转化级别，而是执行几次空循环，如：5 10

锁消除：JIT在编译的时候把不必要的锁去掉；

作业：模式购买飞机票的程序（北京到上海）

